ATARI.RSC
The Atari Developer's Resource

Vol. IV, Issue 1
November 1990 - January 1991

The Bottom Line

Bill Rehbock, Director of Technical Services

Wow... it's been quite a busy year so far! Let me quickly recap the goings-on at Atari in the last two months. MegaSTE and TT computers are currently shipping to key dealers and VARs across the United States at incredibly lucrative prices. The TT030/2-50 (2 megabytes of RAM, 48mb hard drive) has a retail price of only $2399.95, and the TT030/8-80 is priced at only $3799.95. The two and four megabyte versions of the MegaSTE (both include a 48mb hard drive) are $1699.95 and $1849.95, respectively. Profit margins to the dealers have been increased significantly, so street prices will be lower, of course; the two megabyte TT will probably be sold street price for about $2k, the eight megabyte version about $3k, and the MegaSTE will probably be seen for around $1200. I almost forgot to mention the 1040STE... it now retails for $599.95.

We have implemented the *Strategic Partner* dealership program, whereby most stores will purchase all product except TTs through regional distributors such as Pacific Software, Almo Distributing, and Josha Corporation. This will provide the smaller dealers that can't afford large quantities of inventory with a steadier flow of product, much faster shipping, and in most cases "one stop shopping" for hardware *and* software.

The key dealers will grow out of our existing dealer base, primarily by displaying high-end product knowledge, customer service and a professional sales attitude. We will be working very closely and personally with the key dealers, providing market development funds, low or no cost demo hardware, and hopefully with your assistance, software training. This will allow us to coordinate our efforts better as we introduce new products and sales programs.

We are instituting a new warranty/service program shortly that will support our customers and dealers far better than any systems we have used in the past. The new system is streamlined and very customer oriented. It should make it much easier to sign up new dealers and for dealers to sell product. I hope to have the details for you in next month's newsletter.

Speaking of the newsletter, The ATARI.RSC has returned with a new look and feel. We will be including more information pertinent to developing applications for the new product line with your needs in mind. In general, it will have a more "techy" feel, with lots of code fragments and programming examples. We will be including message threads from the ATARI.RSC roundtable, Atari's official developer support platform on GEnie. It is going to be a busy year, and we will do our best to keep you informed of new product developments.

Moving to the future, I am determined to make the current chicken and egg situation in the US market end quickly. We can no longer continue going through the loop of: It's great hardware, but there isn't any software; there isn't any software because there aren't any sales; there aren't any sales because there aren't any dealers; there aren't any dealers because there isn't any product; there isn't any product because it's all going over to Europe where there *is* software and a high demand for the product.

I am currently working with several European and American developers and publishers to bring software to the United States and to make sure that it is properly supported and represented. If you have the ability to provide product support and are

continued on page 12

Inside This Issue

ATARI.RSC
The Resource File

CEO, PRESIDENT
ATARI CORPORATION
Sam Tramiel (408) 745-2324

GENERAL MANAGER
ATARI U.S.
Greg Pratt (408) 745-2349

TECHNICAL DIRECTOR
Bill Rehbock (408) 745-2083

DEVELOPER TECHNICAL SUPPORT
J. Patton (408) 745-2135
Mike Fulton (408) 745-6833
Fax: (408) 745-2094

DEVELOPER ADMINISTRATOR
Gail Johnson (408) 745-2568

SOFTSOURCE ADMINISTRATOR
Dan McNamee (408) 745-2024

CONFIDENTIALITY

The information in this newsletter is confidential. It is for your use in developing products compatible with Atari computers only. You are responsible for protecting the confidentiality of this material in keeping with your Confidentiality Agreement. If you need to reveal some of the information in this newletter, contact Bill Rehbock first to get permission.

Atari Corp.
1196 Borregas Ave.
Sunnyvale, CA 94088

This newsletter was created using Pagestream on the TT030 computer with TTM 194 monochrome monitor and on the MEGA ST4. It was printed using Ultrascript on the Atari SLM804 Laser Printer.

Your Links to the LYNX

Gail Johnson

Many of you Important Developers need a special lift after a hard day's programming. In recognition of this, I am offering the LYNX game system, cartridges, and accessories at reduced prices especially for you. Orders should be sent with a check or money order and a product description to the address given below. Please note that there is a limit of one item each per developer. Have fun!

PART NO.	DESCRIPTION	PRICE
PA2002	LYNX Game System (includes California Games, Comlinx cable, & AC Adapter)	$133.00
PA2020	Blue Lightning	23.50
PA2022	Rampage	23.50
PA2028	Chip's Challenge	23.50
PA2036	Road Blasters	27.00
PA2021	Electro Cop	23.50
PA2035*	Robo-Squash	23.50
PA2024	Gauntlet	27.00
PA2043	Rygar	27.00
PA2029	Slime World	23.50
PA2023	Gates of Zendocon	23.50
PA2026	Xenophobe	23.50
PA2031	Klax	27.00
PA2030	Zarlor Mercenary	23.50
PA2057	Ms. Pac Man	23.50
PA2041	Paperboy	27.00
PAG1200	Power Supply	7.00
PAG3400	Sun Visor/Screen Guard	3.50
PAG3300	Auto Adaptor	14.00
PAG3350*	Carry Case, Large	14.00
PAG3375	Carry Case, Small	10.50

*out of stock at print time

Please include sales tax for your area
(IL add 6.75%; NY add 8%; CA add 7.00%; TX add 8.25%)

Submit orders to:

Atari U.S.Corporation
Post Office Box 3427
Sunnyvale, CA 94088-3427
Attn: Gail Johnson

Atari Softsource

Dan McNamee

Atari Softsource is ready to go at this time, all we need now is your entries. Please, if you haven't done so yet, make your entries NOW!

Doing your entries is not difficult or all that time consuming if you look through my past articles and prepare the information you need before you start. Believe me, I know. I have done more entries, to date, than anyone else. Even if your entry in Softsource only generates one additional sale for you, then it has already paid for itself. Even if you do not have a demo of your product available, do the text entry into the database. Demos are not required, but they will help sell your product.

Until next time...

Programming With a High Neat-O Factor

Bill Rehbock

Since its introduction in 1985, the ST has evolved quite a bit. The capabilities of the machine have been enhanced by improvements from Atari and third-party developers. The graphics and sound capabilities have improved tremendously with the introduction of the TT as well as add-on boards from companies such as Image Systems, Moniterm, Matrix, and Maxon.

As these enhancements come into use, not only does compatibility become an important factor in software design, but making sure that software takes advantage of new system capabilities is crucial. As the Atari computer product line evolves, it becomes essential that one writes with future platforms in mind.

The Moniterm board for the Mega made it apparent that software would have to expect changes in the environment it was running in and act accordingly.

The most common mistake for an application is making assumptions about the current video mode because of the return value of the the Getrez() call. Using Getrez()+2 is valid for opening a virtual workstation with the correct screen fonts, but you should not make any assumptions regarding screen size or color information based on the value returned by the call.

The VDI will always return all of the correct information regarding maximum screen height and width, maximum colors available and number of bitplanes.

To determine the number of columns and rows available in the current system font, you can divide the screen size (returned by the open virtual workstation call) by the size of the system font, which is returned in the first two values from the graf_handle() function.

Ideally, an application should allow the user to set and save his window size, location, and other preferences, but the application has to act accordingly if the user runs the application in a resolution other than what the preferences were saved in.

This will be a semi-regular column in the ATARI.RSC; in future installments, we'll discuss the importance of windows in multi-tasking environments and appl_read() and appl_write() calls.

VDI WORKSTATIONS REDEUX

Mike Fulton

With the recent introduction of the Atari TT030 and FSMGDOS, some developers have discovered that their programs do not function correctly or do not take full advantage of the machine's abilities, such as the TT's new screen modes. In many cases the problem is that that the GEM VDI workstation is not being opened or used correctly. Instead of obtaining information about the display screen through GEM VDI, some programmers have used shortcuts which worked in the past, but which will not function correctly with new machines and software.

For example, with many older programs, if Getrez() doesn't return 0 for ST Low Resolution, or 1 for ST Medium, they assume that the screen must be in ST High Resolution. In the old days, this would work and programmers wouldn't realize they hadn't done things correctly. But with the TT's new video modes, or with a large screen monitor on the ST computers, this method is going to run into problems.

These shortcuts were used to get information about the system because the programmers believed this method was significantly shorter and easier than using the correct method of obtaining the information through GEM VDI. However, most programs using the shortcuts actually gain very little, because they end up doing most of the required GEM VDI calls anyway, but do not take full advantage of the information returned.

In order to make the correct method easier to use, and to provide an extra incentive to avoid using shortcuts, I have created a C language structure to contain all the information for a virtual workstation.

Using a structure for the VDI workstation information has the advantage of giving everything much more useful names without having to resort to keeping track of a bunch of different global variables. For example, isn't something like "screen.numcolors" much easier to understand and remember than "work_out[13]"? Also, this has the advantage of hiding extraneous information when you don't need it.

The best part is, you gain these advantages without giving anything up. Virtually all C compilers are smart enough to generate the same code for a reference to a member of a structure as for a global variable or member of an array. (In fact, some compilers will generate better code for accessing structures than for array members.)

The demonstration program in listing #2 includes the functions necessary to open VDI workstations using our structure. It starts by initializing the program with GEM AES, and then outputs some sample graphics to a screen workstation followed by a printer workstation.

You should use completely separate structures for different devices. Use one to contain information for the screen and another one for the printer, another one for a metafile, and so on. Note how listing #2 has one structure named "screen" and one named "printer". If your output functions need to talk to more than one device, you should consider using a pointer to the current workstation structure, as shown in the output() function in listing #2.

Using a pointer to your current VDI workstation information makes it easier to write the output portion of your programs to work with different GEM devices. The workstation structure includes a device ID field that can be used to determine if the workstation is a screen, printer, or any other device. Your program can check this before doing anything specific to a certain device (like changing the filename for a metafile, for example). For example, at the end of the output() function in listing #2, it checks for device 21 (the printer) to see if it should do a v_updwk() call.

In Listing #2, The top of the program includes some of the standard C header files, and also the new header file VDIWORK.H, which contains the definition of our C structure for the GEM VDI workstation information, as shown in Listing #1.

Next comes a declaration of two workstation structures, "screen" and "printer", which will hold workstation information for these devices.

The open_vwork() function opens a virtual workstation for the current screen resolution, and places the workstation information into the structure which was passed, via a pointer, to the open_vwork() function.

The open_work() function is similar to open_vwork(), except that it opens a physical workstation instead of a virtual one. It also accepts an additional parameter which is used to specify the desired device. This can open printer devices, metafile devices, and so on.

The ext_inquire() function does an GEM VDI vq_extnd() call (extended inquire) to get additional information about the workstation. It is used by open_vwork() and open_work(), and should also be used by your application after doing something which may affect the workstation information (such as loading or unloading fonts).

The remainder of the program is mainly there just to demonstrate the use of the structure and the functions just mentioned. The main() function initializes the program

continued on page 5

VDI Workstations Redeux
(continued from page 4)

with GEM AES, and then tries to open a virtual workstation to the screen using open_vwork(). If this is successful, then it calls the output() function, which just does some simple graphics. Next, it tries to open a printer workstation using open_work(), and if successful, calls output() again to do some graphics, this time to the printer.

Besides the VDI workstation information, the structure also stores the system font size values returned by the AES graf_handle() call. These values can be used, along with the screen resolution, to determine the number of columns and rows of text available.

The sample listings have been tested with Mark Williams C v3.0.9, and with Lattice C v5.0.4. The top of listing #2 includes some #define statements that customize certain parts of the program for different compilers. However, the program should work with little or no modification with just about any C compiler for the ST/TT.

An ARC file containing these listings is available on GENIE in the ATARI.RSC roundtable. The filename is OPENVWRK.ARC. If you have any questions regarding these listings, please leave EMAIL to MIKE-FULTON, or call me at Atari at (408) 745-6833.

See Listing #1 below and Listing #2 on page 6.

VDI Workstations Redeux Listing #1

```
/*VDIWORK.H */
/*by Mike Fulton, ATARI CORP.  */

#ifndef WORD          /* if WORD not already */
#define WORD short    /* defined, then define */
#endif                /* it as a short int, */
                      /* which is a 16-bit size */
                      /* variable in all current */
                      /* ST/TT C compilers */
                      /* including MWC, Alcyon, */
                      /* Lattice, & Aztec */
                      /* If your compiler is */
                      /* different, fix this! */

typedef struct vdiwk{
   WORD  handle, dev_id;     /* Handle, & device */
                             /* ID opened */

   WORD  wchar, hchar, wbox, hbox;  /* Returned */
                                    /* by graf */
                                    /* handle() */

   WORD  xres, yres;       /* Stuff returned by */
                           /* v_opvwk() */
   WORD  noscale;
   WORD  wpixel, hpixel;
   WORD  cheights;
   WORD  linetypes, linewidths;
   WORD  markertypes, markersizes;
   WORD  faces, patterns, hatches, colors;
   WORD  ngdps;
   WORD  cangdps[10];
   WORD  gdpattr[10];
   WORD  cancolor, cantextrot;
   WORD  canfillarea, cancellarray;
   WORD  palette;
   WORD  locators, valuators;
   WORD  choicedevs, stringdevs;
   WORD  wstype;
   WORD  minwchar, minhchar;
   WORD  maxwchar, maxhchar;
   WORD  zero5;
   WORD  maxwline;
   WORD  zero7;
   WORD  minwmark, minhmark;
   WORD  maxwmark, maxhmark;

   WORD  screentype;         /* Stuff returned */
                             /* by  vq_extnd() */
   WORD  bgcolors, textfx;
   WORD  canscale;
   WORD  planes, lut;
   WORD  rops;
   WORD  cancontourfill, textrot;
   WORD  writemodes;
   WORD  inputmodes;
   WORD  textalign, inking, rubberbanding;
   WORD  maxvertices, maxintin;
   WORD  mousebuttons;
   WORD  widestyles, widemodes;
   WORD  reserved[40];
} VDI_Workstation;


/* Array indices for cangdp[] and gdpattr[]
   members of structure */

enum WsGDPs { gdpBAR, gdpARC, gdpPIE, gdpCIRCLE,
              gdpELLIPSE, gdpELLARC, gdpELLPIE,
              gdpRRECT, gdpFRRECT, gdpTEXT };

/* values for gdpattr[] member */

enum WsAttributes {attrPLINE, attrPMARKER,
                   attrTEXT, attrFILL, attrNONE };

/* Type flags for wstype member */

enum WsTypes { wsOUTPUT, wsINPUT, wsINOUT,
               rsRESVD, wsMETAFILE };

/* Flag values for textrot member */

enum WsTextRot { TrNONE, Tr90, TrANY };
```

VDI Workstations Redeux Listing #2

```
/* VWRKDEMO.C by Mike Fulton, ATARI CORP. */
/* Copyright (c) 1990, ATARI CORP.        */


#include <osbind.h>
#include <vdiwork.h>

#define MWC 1           /* Choose your complier */
#define ALCYON_C 0      /* here! These can be */
#define LATTICE_C 0     /* defined in the */
                        /* makefile, if you are */
                        /* using make, but isn't */
                        /* necessary unless you */
                        /* are trying to write */
                        /* portable code */

#if (MWC | ALCYON_C)
#include <gemdefs.h>    /* Mark Williams, Alcyon */
#endif

#if LATTICE_C
#include <aes.h>                /* Lattice C */
#define BUSY_BEE BUSYBEE
#endif

/*****************************************/

VDI_Workstation screen, printer;

short gl_apid, handle,
      contrl[12], intin[256], ptsin[256],
intout[256], ptsout[256];

void ext_inquire(), output();

/*********************************************/
/* Open a virtual workstation for the        */
/* current screen resolution.                */
/* 'dev' - pointer to workstation structure */
/*********************************************/

short
open_vwork( dev )
VDI_Workstation *dev;
{
short i, in[11];

   in[0] = Getrez() + 2;  /* Device to be opened */
   dev->dev_id = in[0];
   for( i = 1; i < 10; in[i++] = 1 );
   in[10] = 2;
   i = graf_handle( &dev->wchar, &dev->hchar,
                    &dev->wbox, &dev->hbox );
   v_opnvwk( in, &i, &dev->xres );
   dev->handle = i;

   if( i )
     ext_inquire( dev );

   return(i);
}

/*********************************************/
/* Open a physical workstation for the       */
/* specified device.                         */
/* 'devno' - device (e.g 21 for printer)     */
/* 'dev' - pointer to workstation structure */
/*********************************************/

short
open_work( devno, dev )
short devno;
VDI_Workstation *dev;
{
short i, in[11];

   in[0] = devno;
   dev->dev_id = devno;
   for( i = 1; i < 10; in[i++] = 1 );
   in[10] = 2;
   i = devno;
   v_opnwk( in, &i, &dev->xres );
   dev->handle = i;
   if( i )
     ext_inquire( dev );

   return(i);
}

void
ext_inquire( dev )
VDI_Workstation *dev;
{
short out[57];

   vq_extnd( dev->handle, 1, out );

/* Get stuff returned by vq_extnd() */

   dev->screentype = out[0];
   dev->bgcolors = out[1];
   dev->textfx = out[2];
   dev->canscale = out[3];
   dev->planes = out[4];
   dev->lut = out[5];
   dev->rops = out[6];
   dev->cancontourfill = out[7];
   dev->textrot = out[8];
   dev->writemodes = out[9];
   dev->inputmodes = out[10];
   dev->textalign = out[11];
   dev->inking = out[12];
   dev->rubberbanding = out[13];
   dev->maxvertices = out[14];
   dev->maxintin = out[15];
   dev->mousebuttons = out[16];
   dev->widestyles = out[17];
   dev->widemodes = out[18];
}

/*****************************************/
/* Do sample graphics to specified device */
/*                                       */
/* The input parameter 'dev' should be a  */
/* pointer to the workstation structure   */
/* for the desired VDI device.            */
/*****************************************/

void
output( dev )
VDI_Workstation *dev;
{
short xsteps, ysteps, clr, pxy[8];

   xsteps = (dev->xres/2) / dev->colors;
   if( ! xsteps )
     xsteps = 1;

   ysteps = (dev->yres/2) / dev->colors;
   if( ! ysteps )
     ysteps = 1;

   vsf_interior( dev->handle, 2 );
   vsf_style( dev->handle, 19 );

   for( clr = 0; clr < dev->colors; clr++ )
     {
      pxy[0] = 0 + (xsteps * clr);
      pxy[1] = 0 + (ysteps * clr);
      pxy[2] = dev->xres - (xsteps * clr);
      pxy[3] = dev->yres - (ysteps * clr);
      vsf_color( dev->handle, clr );
      v_bar( dev->handle, pxy );
     }

   if( dev->dev_id == 21 )    /* If outputing to */
     v_updwk( dev->handle );  /* printer device */
                              /* then update the */
                              /* workstation */
}
```

continued on page 7

VDI Workstations Redeux -- Listing #2
(Continued from page 6)

```
/************************************************/
/* main() initializes the program with GEM AES, */
/* then attempts to open workstations and do    */
/* graphics with the screen, and then with the  */
/* printer device.                              */
/************************************************/


void
main()
{
  gl_apid = appl_init();
  graf_mouse( ARROW, 0L );

  if( open_vwork(&screen) )   /* Open a screen */
    {                         /* workstation */
      graf_mouse( M_OFF, 0L );

/* Do some misc. graphics to this device. */

      output(&screen);
      v_clsvwk( screen.handle );
      graf_mouse( M_ON, 0L );
    }

/* Now do some more graphics with the printer */
/* The results will look a bit different */
/* because of the fewer number of colors */
/* available on the printer. */

    if( open_work( 21, &printer ) )
      {
        graf_mouse( BUSY_BEE, 0L );
        output( &printer );
        v_clswk( printer.handle );
        graf_mouse( ARROW, 0L );
      }

    evnt_keybd();
    appl_exit();
}
```

Portfolio Application Notes

J. Patton

Here are some useful tips and hints discovered by Portfolio developers.

• Using CPU intensive routines will shorten battery life. Avoid using processor loops, rather use hardware timers wherever possible.

• If you need keypresses, use the BIOS. Since the Portfolio will sleep after detecting no keypress through the BIOS after a period of time, addressing the keyboard directly will ignore this mechanism.

• Applications should check that there is sufficient RAM and suggest the ratio to reconfigure for the RAM disk. But it is important to take into account that yours is not the only application and forcing a user to FDISK only for your product may be unacceptable.

• Although we suggest that users back up their RAM disk contents with RAM cards or a parallel or serial interface to a desktop machine, it should not be assumed that they own any of these.

Q&A

Q. How can I prevent the Portfolio from powering down?
A. You need to write a TSR to trap Int 16 Fn 0 calls and return turn these into Int 16 Fn 1 calls until a key comes in.

Q. Is there a way a program can increase the volume of the Portfolio's speaker?
A. No, the only way to increase the volume of Portfolio's speaker is through hardware.

Q. I'm trying to implement multiple header drivers on the Portfolio, but with little success. It works on my PC but not on the Portfolio.
A. The Portfolio's operating system will not handle multiple header drivers, although it will correctly handle single driver headers.

Q. Is there access to the power down/wake up routines (such as the vector the alarm uses)?
A. The alarm uses vector 4Ah, which can be revectored to your own application. However this would prevent the alarms from operating.

Q. How can my application read the Control-S key as it is used in the internal application for searching? I am using getch() now.
A. The getch() call uses DOS function 08h (character input without echo) and doesn't return the keypress. Instead, it pauses and resumes when you hit another key. Function 07h will return unfiltered character input without echo.

New Online for the Portfolio

PBASIC 3.1 - Basic written by B.J. Gleason for the Portfolio.

PGSHOW & utilities - Graphics displayer and tools for the Portfolio written by Don Messerli.

OPTIMIZING 68000 ASSEMBLY LANGUAGE

Ron Cox

When programmers need to get the most out of their code, they often turn to assembly language in order to customize the code and maintain control over the processor. The decision on whether or not assembler can help increase the speed of your program usually lies in the basic algorithm you are employing. Assembler will not add a significant amount of speed to a brain-dead algorithm. That's a fact. However, if you are certain the method you are using to attack the problem is the most efficient available, then it may be helpful to move to assembler for more speed.

One of the first problems with this line of thought is that people assume that just because a program is written in assembler it's as fast as it can get. Not true. Just as programmers write bad C or Pascal code, programmers can write bad or inefficient assembler code. Often the inefficiency stems from not understanding the processors instruction set, and how the processor executes each instruction.

In this article I will attempt to outline some methods which you can use to write faster, smaller assembler code. Some of the things I will discuss are widely known and basic common sense, but hopefully each reader will come away with at least one bit of knowledge they didn't already have. Lets get started, okay?

THE BASICS

Although some of the following tips are very basic, we (me included) often overlook them because they are so simple.

Don't use the 68000 'multiply' group of instructions for multiplying by a constant. If you are multiplying a register by two or four, add the register to itself once or twice, for example:

```
add.l   d0,d0   ; Multiplied by 2
add.l   d0,d0   ; Multiplied by 4
```

If you need to multiply by a multiple of two other than two or four, shift the register left the appropriate number of times, for example:

```
asl.l   #3,d0    ; Multiply d0.l by 8
```

You must use the arithmetic shifts rather than the logical shifts because the latter destroy the sign bit.

Most operations done on a data register set the condition codes, so it is unnecessary to explicitly test the register before doing a conditional branch, for example:

```
     move.l  #10000,d0
again:
     subq.l  #1,d0    ; Subtract 1 from d0
;    tst.l   d0       ; Unnecessary
     bne     again
```

The tst.l operation on a data register uses 4 cycles, so if it were embedded in a loop which executes 10,000 times, you can save 40,000 cycles by removing it. Quite a savings. You may decide that having the tst.l there is good documentation and in that case, leave it, but comment it out as I have done in the example.

When using one of the many branch instructions, there are two modes which may be used, byte and word addressing. In the byte addressing mode, a twos complement 8 bit displacement is added to the PC giving you an effective range of 127 bytes forward or backward. In the word mode, a twos complement 16 bit displacement is added to the PC giving an effective range of 32,767 bytes forward or backward. The savings comes in the form of the size of the instruction, and the time it takes to take or not take the branch. In the previous example, the bne instruction did not specify an addressing mode and most one pass assemblers (like Madmac) will default to word addressing. The instruction works out to be four bytes long, and if the branch is taken consumes 10 cycles, if it is not taken, 12 cycles. Let's rewrite the example so that the branch uses short (byte) addressing:

```
again:
     subq.l  #1,d0   ; Subtract 1 from d0
;    tst.l   d0      ; Documentation
     bne.b   again   ;.b for byte or
                     ; short addressing
```

In this example two bytes are generated for the branch instruction, it uses 10 cycles if taken, 8 if not taken. So in instances where the branch is not taken, the short mode saves 4 cycles over the long, and it always saves 2 bytes of object code. Of course, the deciding factor on which mode to use is how far the branch is. If it is more than 127 bytes forward or back you must use the long mode. Many multi-pass assemblers will optimize branches for you. If you are using Madmac you can ask it to notify you about long branches that may be made into short ones. (Madmac's -s switch reports branches which could be optimized by changing them to short addressing.)

When using address registers remember that word values moved into an address register are sign extended to a long before they are moved.

continued on page 9

Optimizing Assembly Language
(continued from page 8)

Therefore there is no need to explicitly extend the register after the move, for example:

```
    move.w  #$d023,a0
;   ext.l   a0              ; Unnecessary
```

Before moving the value $d023 into a0, the processor sign extends it to a full long word, making the ext.l instruction unnecessary. Leaving it out saves 4 cycles and 2 bytes of object code. Again, you may wish to leave it in there as documentation, but comment it out as I have done above.

It is often necessary to clear a data register and most programmers use the logical choice, clr dn. This is the fastest method of clearing the byte or word of a data register, but if you need to clear the entire 32 bits, use the following:

```
;   clr.l   d0          ; Uses 6 cycles, 2 bytes
    moveq.l #0,d0       ; Uses 4 cycles, 2 bytes
```

THE STACK

The stack is the center piece around which the 68000 operates. Although there are a variety of effective, efficient methods of manipulating the stack, some improvements may still be made.

When calling a subroutine that expects parameters on the stack, it is usually left to the caller to clean the up the stack after the call, and most programmers use the following:

```
    move.l  #buffer,-(sp)   ; Push address of buffer
    move.l  #256,-(sp)      ; read 256 bytes
    move.w  handle,-(sp)    ; from file handle
    move.w  #$3f,-(sp)      ; GEMDOS Fread
    add.l   #12,sp          ; Clean stack
```

A couple of improvements can be made here, but lets start with cleaning the stack. If the value to be added to the stack pointer is 8 or less, use the addq.l instruction. But, as above, if the value is greater than 8, use the following:

```
...
    move.w  #$3f,-(sp)          ; GEMDOS Fread
    lea     12(sp),sp           ; faster stack clean
```

Not only does using load effective address save a couple of cycles, it saves 2 bytes of object code as well.

Another improvement can be made in the way parameters are pushed onto the stack. When most programmers move an address onto the stack they use a move instruction. But, the 68000 includes an instruction that automatically calculates the effective address of an operand and pushes it on the stack.

This instruction is pea, or "push effective address", for example:

```
;   move.l  #buffer,-(sp)       ; Change to
    pea     buffer              ; Faster
```

Although the push effective address instruction will not save you any object code, it will save you 8 cycles, and when placed in a loop that can be a significant savings. This brings us to our next and last optimization.

When making a subroutine call (with parameters on the stack), there are two situations where time and object code can be saved. In the first example, when several calls are made in sequence, stack cleaning can be put off until after all the calls are made, for example:

```
    move.w  #66,-(sp)       ; Word to convert
    jsr     _toascii        ; To ascii (Don't clean
                            ; stack yet!)
    move.w  d0,-(sp)        ; Returned ascii value
    move.w  #2,-(sp)        ; GEMDOS Conout
    trap    #1              ; Call GEMDOS
    addq.l  #6,sp           ; Clean stack from all
                            ; previous calls (3 words)
```

The other situation is when your code is in a loop. If the loop is known to execute n times, and n is small enough (or if your stack is big enough), stack cleaning can be put off until the loop is done executing, for example:

```
    move.w  #25,d1          ; Execute 25 times
    lea     buffer,a0       ; place to put results

loop:
    move.w  d1,-(sp)        ; Word to convert to ascii
    jsr     _toascii        ; Would normally clean
                            ; stack here
    move.b  d0,(a0)+        ; Copy to character to
                            ; buffer
    subq.w  #1,d1           ; Decrement counter
    bne.s   loop            ; Zero yet??

                            ; Clean stack here
    lea     50(sp),sp       ; 25 iterations times 2
                            ; bytes pushed for each
                            ; iteration
```

This is a somewhat obscure optimization, but it saves a great deal of time and could be used where absolute speed is desired. One more note before concluding, notice I used the load effective address instruction to get the address of buffer into a0. This saves 8 cycles over its move.l counterpart.

THAT'S ALL, FOLKS

This article is not meant to be the final word on optimizing assembler, but rather a testimony to the fact that indeed assembler can be written faster and smaller if care is taken to use the proper instructions.

(continued on page 10)

Optimizing Assembly Language
(continued from page 9)

This is an incomplete list of optimizing techniques, but one which I feel will cover most of the situations which programmers come across.

I have taken great care to insure that the information contained herein is as accurate as possible, but I am sure that some bug crawled in during the middle of the night and munched on part(s) of the text, and I cannot accept any responsibility for the damage done :-). If you have any questions or just want to point out corrections that need to be made, I can be reached on GEnie at "R.J.COX". Happy programming!

Bibliography:

Morten, Mike "68000 Tricks and Traps". BYTE. September 1986.

Veronis, Andrew M. The 68000 Microprocessor New York, NY: Van Nostrand Reinhold, 1988

ABOUT THE AUTHOR:
Ron Cox has been an Atari Developer for over 3 years. He is currently pursuing a Bachelor degree in Computer Information Systems while working on a project with Darin Wayrynen (for the Atari).

Line-A Support In Future Video Modes

Mike Fulton

At this time, it has been decided that the Line-A low level graphics interface will not be officially supported by Atari in new video modes available on the TT030 and any future machines.

Line-A will still be supported in all ST-compatible resolutions, so existing programs should still work in those modes. However, in order to be compatible with new video modes, and to take full advantage of new features, programs currently under development should use GEM VDI exclusively and avoid using Line-A.

Almost all of the information and graphics commands available through Line-A can be easily accessed or done using GEM VDI. Most programs using Line-A can switch to using GEM VDI without any problems. If you've always used Line-A for your graphics, and never even tried using GEM VDI, then see the article "VDI Workstations Redeux" in this newsletter for more on using GEM VDI to obtain information about the current screen mode.

There are several reasons why Line-A support will not be extended to new video modes. However, it should be restated that Line-A is an interface to low-level portions of GEM VDI, and not some separate entity that VDI happens to use. It is a way to access some of VDI's global system variables and low-level graphics primatives.

The VDI has already been updated to support the new video modes of the TT030. It will have to be updated again to support new video modes that will be available on future machines to come out down the line. At least some of these video modes will use screen memory layouts andimproved features which are drastically different from what is currently use, and Line-A is quite simply not setup to support them.

Existing programs using Line-A are much less likely, as a general rule, to work correctly on the new video modes of the TT030 than are programs which use GEM VDI. While programs which use GEM VDI aren't always written correctly to take full advantage of all the features of new video modes, they generally do function. Furthermore, the basic method of using GEM VDI doesn't have to change to support new video modes. However, if Line-A were to be updated to support these new video modes, the method of making calls would have to change significantly, so existing programs would not work in the new modes anyways.

For those who are concerned about a speed difference between Line-A and GEM VDI, consider that the machines that will have these new video modes will be much, much faster than the original Atari ST. There are no significant performance differences between Line-A and GEM VDI.

Atari recognizes that there are certain bits of information available through Line-A which are not easily accessed using GEM VDI by certain categories of programs like TSR utilities, most notably the current mouse position. Because of this, I'm asking developers for the following: examine your programs and the Line-A variable table, and construct a list of which Line-A variables you consider to be absolutely essential. Please include detailed justification of why you thinkthese variables are essential and why you cannot use GEM VDI for this purpose. We will take this list and see what can be done in accomodating it.

We don't make any promises, and this does not apply to Line-A calls that perform actual output. We are only talking about variables in the Line-A data table.

Please mail your suggestions to my attention at Atari Corp., or you can send private EMAIL on GEnie to MIKE-FULTON, or on Compuserve to 75300,1141.

In the meantime, if you are writing a program, and you feel you absolutely must use Line-A for something, please contact Atari Developer Support for advice and additional information.

ATARI Developer News

ATARI.RSC Staff

Welcome, James Grunke

Atari would like to introduce James Grunke to all of our MIDI developers. As of January 1, James has joined Atari U.S. as the new MIDI Marketing Director. James comes to us after three years with Brother Records, for whom he served as a Programmer, Technician, and Performer.

If you have any questions or ideas about marketing your MIDI products for the Atari, you can contact James at the address and phone number below.

James Grunke
Atari U.S. Corp.
MIDI Marketing Director
1196 Borregas Ave.
Sunnyvale, CA 94086

(408) 745-4966
(408) 745-2088 - fax

Atari Increases Developer Support Staff

Atari would also like to introduce Mike Fulton to all of our software developers. Mike has joined Atari's Developer Support staff after three years at Neocept, where he served as a programmer and technical support representative.

If developers have questions regarding technical matters, programming, and so forth, or even if you're just not sure who else to ask about something, then you should contact either Mike Fulton or J. Patton for Developer Support. For information regarding equipment loans or marketing, please contact Bill Rehbock.

Developer Support is also available on the GEnie telecommunications system through the ATARI.RSC roundtable. This is a specialized roundtable just for Atari devlopers. In addition to J. Patton and Mike Fulton, the roundtable staff includes Atari programmers John Townsend and Ken Badertscher.

Atari Developer Support can be reached at address and phone numbers given on page 2 of the ATARI.RSC newsletter.

ATARI.RSC staff GEnie Mail Addresses

Ken Badertscher = K.BAD
John Townsend = TOWNS
J. Patton = ATARIDEV
Bill Rehbock = B.REHBOCK
Mike Fulton = MIKE-FULTON

To send mail to all of the above, you can use the ATARI.RSC staff mail address: RSC$

Confidentiality: What Exactly Can You Talk About?

Some developers have expressed some concern and confusion about exactly which materials from the devlopement package are covered by the non-disclosure agreement, and which materials they are free to discuss with regular users.

Anything specifically marked or described as "Confidential Information" is strictly off-limits for outside discussion with non-developers (and even with other developers if so noted).

Anything in the various "Release Notes" documents that Atari has sent out is OK to talk about. Documents in this category include:

Rainbow TOS Release Notes
STE TOS Release Notes
TT030 TOS Release Notes
AHDI 3.00 Release Notes

Anything included in a document that is not marked "Release Notes" should be treated as confidential even if not marked as such. This includes documents like the "TT030 Companion" which some TT developers received. (This was an early version of the TT030 TOS Release Notes document.)

Of course, anything that's been published in one of the various ST programming reference guides, like from Abacus or Compute! Books, is also OK to talk about. When in doubt, please feel free to call developer support and ask.

New Online In ATARI.RSC

CPXDOC.ARC - Documentation and sample files and source code for writing CPX modules for the new eXtended CONTROLpanel.

NEW_LANG.ARC - Utilities from the TT030 Language Disk, including Atari Hard Disk Utilities v4.02, XCONTROL.ACC with CPX files, Mouse Accelerator, Atari Laser Printer Utilities, and more.

Atari's Latest Laser: The Atari SLM605

Atari's latest entry in the personal laser printer market, the Atari SLM605, began shipping worldwide during the fourth quarter of last year. Like its older sibling the Atari SLM804, the new SLM605 operates using the same write-to-white electrophotographic laser technology at a print resolution of 300 by 300 dots per inch.

The Atari SLM605 prints at 6 pages per minute and includes such refinements as a sleeker physical profile, straighter paper path, darker single-pixel printing, decreased stand-by power, quieter operating noise, and a lower list price: $1295.

The SLM605 uses the TEC Model LB-1305 laser print engine, the same engine used by over a half-dozen laser printer manufacturers including Epson (which means that toner and drum supplies are easier to find).

The page printer interface to the Atari SLM605 is identical to the interface on the older SLM804 laser printer controller (in fact, Atari is shipping initial units with the SLMC804 controller). Printer drivers written for the SLM804 will work with the SLM605 without any modifications.

continued on page 12

Atari Developer News
(Continued from page 11)

For Atari TT030 owners, new versions of the GDOS printer driver (SLM.SYS) and Diablo emulator can be found on the TT030 Language Disk included with the workstation. The modifications made to the drivers are of benefit only to Atari TT030 users.

Atari will continue to develop and introduce laser printers following the current market trend of lower cost and higher resolution. Given Atari's commitment to virtual device interfaces (such as the page printer interface), developers of device-driver software are assured of compatibility with each new laser printer introduction.

The Bottom Line
(continued from page 1)

interested in representing a specific European product line, or have written (or partially written) a "slam dunk" application for the ST/TT, please contact me before the Hanover show in March. GEnie Mail address: B.REHBOCK
Fax: (408)745-2088

Some of the new operating system utilities are now available to developers. The eXtensible CONTROL panel documentation is being shipped with this newsletter. Please examine the final specification and bindings closely, it is a good example of operating system extensions and new programming tools that we will be supplying in the very near future. FSMGDOS has already been sent to key GDOS-oriented developers. Due to "paperwork" obligations regarding FSM and its fonts, the FSMGDOS development materials must be distributed on a by-request basis. The most expedient method, of course is by GEnie E-Mail or by fax.

I encourage ***everyone*** to write their applications with GDOS in mind. It is a remarkably programmer-friendly system, and unlike the old bit-mapped GDOS, you will not have to pay any license fees for distribution rights, as it will be shipping with all new computers and shall be made available as a system software upgrade to existing systems. Everyone that has seen FSM in action has been amazed at how much faster it is than Adobe Type Manager on the Mac.

The upcoming several months are going to move quickly, and I truly believe that the future looks much brighter than it has in years. I look forward to a long and profitable relationship with all of you.

Atari Corporation
1196 Borregas Ave.
Sunnyvale, CA 94086

ATARI.RSC
The Developer's Resource